

Nro. ord.	Apellido y nombre	L.U.	#hojas

ORGANIZACIÓN DEL COMPUTADOR I - Parcial

Primer Cuatrimestre 2020

Ej.1	Ej.2	Ej.3	Nota

Corrector:

Aclaraciones

- Anote apellido, nombre, LU en *todos* los archivos entregados.
- Cada ejercicio se califica con Muy Bien (MB), Bien (B), Regular (R) o Mal (M). La división de los ejercicios en incisos es meramente orientativa. Los ejercicios se califican globalmente.
- **Importante:** Justifique sus respuestas. Las soluciones a ejercicios de la práctica que se utilicen deben ser incluidas en el examen.
- El parcial se aprueba teniendo por lo menos dos ejercicios Bien, y uno Regular.
- El link de entrega es: <https://bit.ly/parcial0rga1>. Ante cualquier problema pueden subirlo también al campus.

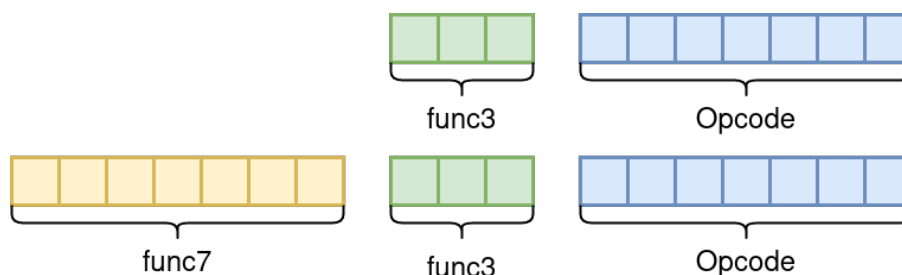
Introducción

Una máquina inspirada en la arquitectura RISC-V utiliza buses de datos y direcciones de 32 bits, memoria direccionable a byte y opera con aritmética en complemento a 2.

Tiene 32 registros **r0-r31** de los cuales **r0** es un registro que siempre contiene el valor cero (es decir, no se puede escribir).

Las operaciones se ejecutan siempre sobre los registros, y el acceso a memoria se realiza solamente mediante las instrucciones **LW** (*Load Word*) y **SW** (*Store Word*). Las instrucciones no generan ni utilizan flags y tienen hasta tres operandos.

Para optimizar la implementación, el código de operación puede tener dos longitudes distintas y se encuentra dividido en tres campos según la siguiente figura:



De esta forma, instrucciones similares comparten el campo *opcode* y se diferencian por los campos *func3* ó *func7*.

El formato de instrucción es:

Formato	Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Registro	func7																																
Inmediato	imm[11:0]																																
Store	imm[11:5]																																
Branch	imm[12]																																

Donde:

- **rs1**: registro fuente 1.
- **rs2**: registro fuente 2.
- **rd**: registro destino.
- El bit **imm[0]** es interpretado siempre como 0 para el formato de instrucción B (notar que dicho bit no está incluido en la instrucción).

Ejemplos de cada tipo:

```
add a2, a1, a0      ; (R) a2 = a1 + a0
addi a1, a0, 0x010  ; (I) a1 = a0 + 0x00000010
lw a2, 4(a1)        ; (I) a2 = mem[a1 + 4]
sw a1, 8(a0)        ; (S) mem[a0 + 8] = a1
sw a1, a0, 8        ; ídem anterior, otra sintaxis
bge a1, a0, 12       ; (B) si (a1 >= a0): pc += 12
```

La máquina posee el siguiente set de instrucciones:

Instrucción	opcode	funct3	funct7	Formato	Efecto
ADD	0110011	000	0000000	R	$rd \leftarrow rs1 + rs2$
SUB	0110011	000	0100000	R	$rd \leftarrow rs1 - rs2$
OR	0110011	110	0000000	R	$rd \leftarrow rs1 \mid rs2$
AND	0110011	111	0000000	R	$rd \leftarrow rs1 \& rs2$
XOR	0110011	100	0000000	R	$rd \leftarrow rs1 \oplus rs2$
SLT	0110011	010	0000000	R	$if(rs1 < rs2) : rd \leftarrow 1, \text{ else } rd \leftarrow 0$
SLTU	0110011	011	0000000	R	$if(rs1 <_{unsigned} rs2) : rd \leftarrow 1, \text{ else } rd \leftarrow 0$
LW	0000011	010	—	I	$rd \leftarrow mem[rs1 + sign_extend(offset)]$
ADDI	0010011	000	—	I	$rd \leftarrow rs1 + sign_extend(imm)$
ORI	0010011	110	—	I	$rd \leftarrow rs1 \mid sign_extend(imm)$
ANDI	0010011	111	—	I	$rd \leftarrow rs1 \& sign_extend(imm)$
XORI	0010011	100	—	I	$rd \leftarrow rs1 \oplus sign_extend(imm)$
SLTI	0010011	010	—	I	$if(rs1 < sign_extend(imm)) : rd \leftarrow 1, \text{ else } rd \leftarrow 0$
SW	0100011	010	—	S	$mem[rs1 + sign_extend(offset)] \leftarrow rs2$
BEQ	1100011	000	—	B	$if(rs1 == rs2) : pc \leftarrow pc + sign_extend(offset)$
BNE	1100011	001	—	B	$if(rs1 \neq rs2) : pc \leftarrow pc + sign_extend(offset)$
BLT	1100011	100	—	B	$if(rs1 < rs2) : pc \leftarrow pc + sign_extend(offset)$
BGE	1100011	101	—	B	$if(rs1 \geq rs2) : pc \leftarrow pc + sign_extend(offset)$

Notas:

- Las instrucciones de tipo B se refieren a *Branches*, que conceptualmente para nosotros representan saltos. Es decir, BNE se interpreta como “Salto si no son iguales”.
- La notación `sign_extend(X)` representa la extensión de signo de X a la cantidad de bits correspondientes.

Ejercicios

Ejercicio 1

En base a la máquina mostrada en la introducción:

a. Determinar:

- El tamaño del PC, el tamaño máximo de la memoria y la cantidad de direcciones de memoria.
- ¿Cuál es el número máximo que podemos sumar a un registro a través del direccionamiento inmediato?

b. Responder:

- Para cada parte de la planilla de seguimiento de la máquina ORGA1, justificar si es necesaria o no cada celda en una máquina tipo RISC-V. Detallar, de ser necesario, nuevas celdas a agregar para poder realizar el seguimiento.
- Modificar la planilla de seguimiento de la máquina ORGA1 según las decisiones tomadas en el punto anterior y realizar el seguimiento de la máquina RISC-V usando la memoria presentada a continuación (siete instrucciones). Se sabe que el PC y todos los registros se inician en cero. La memoria también comienza en cero, salvo en las posiciones que se indican:

Dir:	0x00000000	0x00000004	0x00000008	0x0000000C	0x00000010	0x00000014
Cont:	0x00158613	0x00164693	0x00D65463	0x00068733	0x00D620A3	0xFE0688E3

Ayuda: la planilla de la máquina ORGA1 se puede encontrar en:

<https://bit.ly/planillaOrga1>

Para pasar de hexadecimal a binario se puede utilizar la siguiente línea de Python:

```
bin(int("DEADBEEF", 16))[2:].zfill(32)
```

donde DEADBEEF es la palabra en hexadecimal a convertir, 16 es la base (hexadecimal en nuestro caso) y 32 es la cantidad de bits.

- Para programar: Escribir el código assembler necesario para sumar dos números enteros A y B de 64 bits almacenados de la siguiente forma:

- A = parte alta en $r13$, parte baja en $r12$
- B = parte alta en $r15$, parte baja en $r14$

Almacenar el resultado en dos registros (por ejemplo $r10$, $r11$).

Ayuda: calcular el carry de las partes bajas. Considerar el uso de la instrucción SLTU.

Ejercicio 2

- Realizar el circuito simplificado del componente *BRANCH*, encargado de decidir si se ejecuta o no un salto condicional. Implementaremos sólo la instrucción BGE.
En este caso el circuito toma dos valores A y B (de 4 bits y sin signo, por simplicidad) y devuelve un bit 'b' que vale '1' en el caso que $A \geq B$ y '0' para cualquier otro caso.

Ayuda: resolver primero el caso de igualdad para cada dígito, y luego para la desigualdad utilizar los resultados parciales del caso anterior.

- Realizar el camino de datos (*Datapath*) de una microarquitectura para la máquina en cuestión, que soporte la ejecución de las instrucciones enumeradas en el ejercicio 1. Recordar indicar el tamaño de cada registro, de los buses internos y externos, las señales de cada componente, y justificar la utilización de cada componente elegido y cada decisión tomada. Utilizar los siguientes componentes:

- Una única ALU que realiza las operaciones.
- Un único controlador de memoria (para las instrucciones *SW* y *LW*).
- Extensores de signo. Multiplexores. Incrementador por 4.
- Una unidad de *Decode*
- El componente del apartado anterior (de no haberlo realizado se puede suponer hecho).
Se puede considerar de 32 bits y que implementa todas las instrucciones B.
- Cualquier otro componente que considere necesario.

- Escriba el microprograma que realiza la parte de ejecución del ciclo de instrucción de las siguientes instrucciones:

I. XOR

II. BGE

III. ADDI

Ejercicio 3

En un sistema que utiliza la máquina propuesta en la introducción (inspirada en la arquitectura RISC-V), un sensor de temperatura está mapeado a la posición de memoria 0x00000800. La medición la realiza con 12 bits, abarcando el rango de -30°C a 97°C siendo 0x000 el valor más bajo y 0xFFFF el más alto.

Se quiere controlar el encendido de un refrigerador en función de la temperatura. El mismo se controla con 4 bits (por seguridad) mapeados a los bits menos significativos de la posición de memoria 0x00000801. Si los 4 bits están en '1' el refrigerador se enciende, si están en '0' se apaga. Cualquier otro estado no produce cambios.

- Escribir el pseudo-código y assembler de un programa que controle el encendido/apagado del refrigerador en base a dos umbrales: uno superior para el encendido a 40°C y uno inferior para el apagado a 0°C (este mecanismo es conocido como histéresis).
Codificar los umbrales como constantes en el código y leer la temperatura a través de polling.
- Si ahora el sensor funciona interrumpiendo a la máquina cada vez que la temperatura baja de 0°C , modificar el programa anterior reescribiendo la rutina principal y agregando la rutina de atención de interrupciones.

Ayuda: obviar el control de encendido y apagado de la interrupción así como la preservación de registros. Suponer que esto está controlado automáticamente.